D'un plan optimal parallèle vers un plan optimal séquentiel

Stéphane Grandcolas et Cyril Pain-Barre

LSIS – UMR CNRS 6168
Domaine Universitaire de Saint-Jérôme
Avenue Escadrille Normandie-Niemen
F-13397 Marseille Cedex 20 - France
{stephane.grandcolas,cyril.pain-barre}@lsis.org

Résumé

En planification on distingue les plans optimaux en le nombre d'étapes (plans parallèles) et les plans optimaux en le nombre d'actions (plans séquentiels). Il est généralement admis que le calcul d'un plan séquentiel est plus couteux que le calcul d'un plan parallèle. Büttner et Rintanen ont proposé une procédure de recherche qui calcule des plans dont le nombre d'étapes est fixé et le nombre d'actions minimal. Cette procédure est utilisée pour le calcul d'un plan séquentiel optimal partant d'un plan parallèle optimal. Nous décrivons dans cet article une approche de ce type, développée à partir du système de planification FDP. L'idée consiste à maintenir deux structures, l'une représentant le plan parallèle et l'autre le plan séquentiel, en répercutant les choix effectués pendant la recherche simultanément dans les deux structures. Les techniques développées dans FDP pour le calcul de plans séquentiels ou de plans parallèles permettent de détecter les échecs dans les deux structures. Les résultats expérimentaux montrent que cette approche est très compétitive comparée au calcul de plans séquentiels optimaux obtenus avec FDP.

1 Introduction

Lors de la compétition de planificateurs IPC5 en 2006 un nouveau planificateur, FDP [4], est apparu. Ce planificateur a la particularité de produire des plans séquentiels optimaux. On admet généralement que le calcul de tels plans est plus coûteux que le calcul de plans parallèles optimaux. Les plans parallèles ont moins d'étapes, et sont donc plus *contraints*, ce qui tend à aider la recherche. Comme beaucoup d'autres planificateurs optimaux la preuve d'optimalité vient de la démonstration qu'il n'existe pas de plans en moins d'étapes ou d'actions. Ainsi le système cherche des plans en une étape, puis deux,... tant qu'un plan solution n'est pas trouvé et qu'une borne sur le nombre d'étapes ou d'actions n'est pas atteinte. La simplicité du planificateur FDP vient du fait qu'il implémente sa propre procédure de recherche, de type recherche en avant en profondeur : la procédure tente de toutes les façons possibles d'étendre le plan partiel courant. Le principe général est donc d'effectuer des recherches en profondeurs bornées en augmentant la borne itérativement (IDDFS [6] : Iterative Deepening Depth First Search).

FDP a obtenu des résultats honorables dans la catégorie des planificateurs optimaux (la plupart produisent des plans parallèles). Ce succès tient beaucoup à l'efficacité de la procédure de recherche qui utilise de nombreuses techniques pour éviter des traitements redondants de différents types. Notamment, un ordre *a priori* sur les actions, similaire à l'*élagage de la commutativité* [5], permet d'éviter la construction de séquences d'actions redondantes, ce qui est inutile lors de la recherche de plans parallèles. La procédure fait aussi appel à des évaluations grossières du nombre d'actions pour atteindre les buts qui sont utilisées pour abandonner la recherche de façon précoce. Enfin les états dont l'invalidité est démontrée sont mémorisés dans une table de hashage, afin de ne pas les traiter à nouveau lors d'une rencontre future.

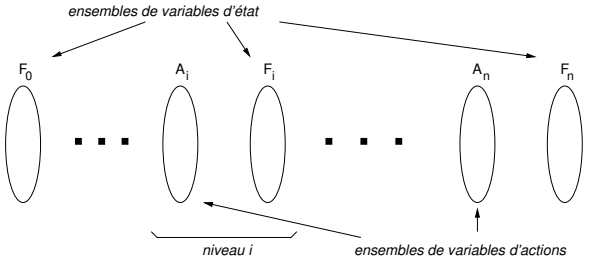
S'il existe peu de travaux sur la recherche de plans séquentiels optimaux, Büttner et Rintanen [2] ont développé un système pour ce calcul à partir d'un plan parallèle optimal. Leur méthode consiste à effectuer des recherches successives à partir de ce plan, en di-

minuant le nombre d'actions quite à accroître, s'il n'y a pas de solution, le nombre d'étapes. La recherche se termine lorsque le nombre d'étapes et le nombre d'actions sont égaux. On espère de cette façon accélérer la recherche du fait que le problème est très contraint d'une part, et du fait qu'on a à traiter des problèmes plutôt satisfiables d'autre part. La procédure de recherche qui est utilisée dans ce cadre détermine si il existe un plan solution en m étapes et contenant au plus n actions. Remarquons que cette procédure en soit est originale et permet par exemple de calculer des plans optimaux contraints sur leur nombre d'actions (en augmentant le nombre d'étapes tant qu'aucune solution n'est rencontrée) ou encore des plans optimaux contraints en leur nombre d'étapes (dans ce cas en augmentant le nombre d'actions). Nous proposons dans cet article, en nous appuyant sur les travaux de G. Gabriel, S. Grandcolas et C. Pain-Barre sur la recherche de plans séquentiels optimaux [4] et de plans parallèles optimaux [3], une approche originale pour le calcul de plans optimaux contraints en leur nombre d'actions ou d'étapes. Le principe consiste à maintenir parallèlement deux structures représentant le plan séquentiel en construction et le plan parallèle correspondant. Les techniques développées dans FDP sont utilisées ici dans les deux structures. Pendant la recherche l'une ou l'autre provoque des échecs. Nous avons implémenté cette approche et l'avons comparée aux résultats obtenus par FDP [1].

Dans la première partie nous revenons brièvement sur le planificateur FDP dans sa version séquentielle et sa version parallèle et sur les FDP-structures qu'il utilise. Ensuite nous présentons la procédure de recherche d'un plan de m étapes et n actions. La troisième partie est consacrée au calcul d'un plan séquentiel optimal à partir d'un plan parallèle optimal comme l'ont proposé Büttner et Rintanen. Enfin dans la quatrième partie sont présentés les résultats expérimentaux que nous avons obtenus.

2 FDP-structures

Un problème de planification $\mathcal{P}$ est un triplet (I, G, A) où I est l'état initial, G est l'ensemble des buts à satisfaire et A est l'ensemble des actions. De $\mathcal{P}$, on en déduit l'ensemble F des fluents du problème. Une action a de A a une précondition $pre(a)$ et des effets $\mathit{eff}(a)$. Une action ajoute un fluent f si $f \in \mathit{eff}(a)$. Elle le supprime si $\neg f \in \mathit{eff}(a)$.
Fondamentalement, le cœur de FDP est une fonction de décision répondant à la question : existe-t-il une solution de longueur n à un problème $\mathcal{P}$? En planification séquentielle, n est le nombre d'actions que doit comporter la solution. En planification parallèle, c'est son nombre d'étapes, où chaque étape comporte un ensemble non vide d'actions compatibles. Pour y répondre, FDP utilise une structure de type CSP de taille n, appelée *fdp-structure*, qui représente un ensemble de plans potentiellement valides de longueur n. La recherche consiste alors à supprimer ou fixer des actions dans la structure, jusqu'à ce qu'elle ne contienne plus qu'un plan valide satisfaisant G, ou qu'un échec soit démontré.
La *fdp-structure* est composée d'un ensemble de variables soumises à des contraintes, partitionnées en ensembles de variables d'état et en ensembles de variables d'actions. Une fdp-structure de longueur n comporte $n+1$ ensembles de variables d'état et n ensembles de variables d'action. Elle peut être vue comme un graphe nivelé, similaire à celui de Graphplan [1], où un niveau i comporte un ensemble de variables d'action A_i et un ensemble de variables d'état F_i :



L'ensemble F_i code l'état après i étapes : pour chaque fluent $f \in F$, une [2] variable d'état f_i indique s'il est vrai, faux ou indéfini. S'il est indéfini, l'état est lui-même partiellement défini. Lorsqu'aucune variable de F_i n'est indéfinie, alors F_i est un état. L'état initial est codé par F_0 et est totalement défini, selon l'hypothèse du monde clos. L'ensemble A_i code les actions de l'étape i. Il diffère selon que la planification est séquentielle ou parallèle. En planification séquentielle, une seule action est autorisée par étape : A_i se réduit alors à une seule variable a_i dont le domaine est l'ensemble A des actions du problème. En planification parallèle, plusieurs actions peuvent être exécutées dans une étape, et A_i contient autant de variables d'actions que d'actions, chacune indiquant si l'action correspondante est fixée, supprimée ou possible à cette étape. On définit ainsi deux fdp-structures : $S_{seq}(n) = (\{F_0^{seq}, \ldots, F_n^{seq}\}, \{A_1^{seq}, \ldots, A_n^{seq}\})$ pour la planification séquentielle et $S_{par}(n) = (\{F_0^{par}, \ldots, F_n^{par}\}, \{A_1^{par}, \ldots, A_n^{par}\})$ pour la planification parallèle.

1. Aucun exécutable de Büttner et Rintanen n'étant disponible, nous n'avons pas comparé notre méthode à la leur.

2. En réalité, nous utilisons deux variables : f_i pour le f et $\neg f_i$ pour $\neg f$. Nous ferons le plus possible abstraction de la seconde pour ne pas surcharger inutilement l'article.

Avant de commencer la recherche d'une solution de longueur n, toutes les variables d'état ont pour domaine $\{vrai, faux\}$ (le fluent correspondant est indéfini). En planification séquentielle, toutes les variables d'actions ont pour domaine l'ensemble des actions du problème. En planification parallèle, leur domaine est $\{vrai, faux\}$. Dans les deux cas, toute action est donc possible à chaque étape. Ensuite, les domaines des variables de F_0 sont réduits pour que F_0 corresponde à l'état initial I. De même, ceux de certaines variables de F_n sont réduits pour correspondre aux buts G. La réduction du domaine d'une variable d'état s'apparente à la suppression d'un littéral : enlever $vrai$ pour f_i signifie que f est supprimé de l'étape i, ne pouvant y être vrai. Par abus de langage, nous dirons que f_i est vrai (faux) si $vrai$ ($faux$) est la seule valeur de son domaine.
Ces "suppressions" rendent incohérentes des valeurs dans le domaine d'autres variables, par propagation. En effet, les variables d'une fdp-structure sont soumises à des contraintes implicites, inhérentes à la planification. Par exemple, si un littéral est supprimé à une étape, alors les actions qui l'ont en précondition peuvent être supprimées de l'étape qui suit. De même, les actions qui ajoutent ce littéral peuvent être supprimées de l'étape qui précède. Cette propagation, similaire au maintien de la consistance d'arc [7], est assurée par la fonction MakeConsistent de FDP. Elle est appelée à la suite de la réduction d'un domaine d'une variable : à l'initialisation de la structure, puis pendant la recherche lorsqu'un choix est opéré (ce qui se traduit par la réduction d'un domaine). Cette fonction rend cohérente une fdp-structure en supprimant des littéraux ou des actions lorsque ceux-ci ne sont plus cohérents, ainsi que les littéraux et les actions que ces suppressions ont rendu incohérents. Elle peut échouer, notamment lorsqu'un littéral et son opposé sont incohérents (tous les deux vrais ou tous les deux faux). Une cause d'échec en planification séquentielle est lorsqu'aucune action n'est possible à une étape. Dans ce cas, la procédure de recherche doit remettre en cause le choix qui a conduit à l'échec.

MakeConsistent utilise les règles suivantes pour déterminer si un littéral ou une action est incohérent. Une seule règle ne s'applique qu'à la planification parallèle et est marquée par (par). Une autre ne s'applique qu'à la planification séquentielle. Elle est marquée par (seq). Par simplicité, on notera A_i l'ensemble des actions possibles (et non choisies) à l'étape i, et X_i l'ensemble des actions fixées à l'étape i. En planification séquentielle, X_i est vide ou A_i est vide et X_i ne peut être qu'un singleton.

Soit un fluent $f \in F$ et f_i sa variable associée à l'étape i. La valeur $vrai$ (resp. $faux$) pour f_i est incohérente si l'une des conditions suivantes est satisfaite :

- $i > 0$ et $\neg f_i$ est vrai (resp. faux)
- $i > 0$ et $\exists a \in X_i$ t.q. $\neg f \in \mathit{eff}(a)$ (resp. $f \in \mathit{eff}(a)$)
- $i < n$ et $\exists a \in X_{i+1}$ t.q. $\neg f \in \mathit{pre}(a)$ (resp. $f \in \mathit{pre}(a)$)
- $i > 0, X_i = \emptyset$ et $\forall a \in A_i,\ \neg f \in \mathit{eff}(a)$ (resp. $f \in \mathit{eff}(a)$)
- $i < n, X_{i+1} = \emptyset$ et $\forall a \in A_{i+1},\ \neg f \in \mathit{pre}(a)$ (resp. $f \in \mathit{pre}(a)$)
- $i > 0, f_{i-1}$ est faux (resp. vrai) et $\forall a \in X_i \cup A_i,\ f \notin \mathit{eff}(a)$ (resp. $\neg f \notin \mathit{eff}(a)$)
- $i < n, f_{i+1}$ est faux (resp. vrai) et $\forall a \in X_{i+1} \cup A_{i+1},\ \neg f \notin \mathit{eff}(a)$ (resp. $f \notin \mathit{eff}(a)$)
- $i > 0, \exists f'_i$ t.q. f'_i est vrai et mutex(f_i, f'_i, i) (resp. mutex($\neg f_i$, f'_i, i)) ou f'_i est faux et mutex(f, $\neg f'_i$, i) (resp. mutex($\neg f_i$, $\neg f'_i$, i)

Une action a est incohérente à une étape $i > 0$ si l'une des conditions suivantes est satisfaite :

- $\exists f \in \mathit{pre}(a)$, t.q. f_{i-1} est faux
- $\exists f \in \mathit{eff}(a)$, t.q. f_{i+1} est faux
- (par) $\exists a' \in X_i$ t.q. mutex(a, a', i)
- (seq) $\exists f \in F$ t.q. f_i est faux (resp. vrai), f_{i+1} est vrai (resp. faux), et $f \notin \mathit{eff}(a)$ (resp. $\neg f \notin \mathit{eff}(a)$)
- $\exists f \in \mathit{pre}(a)$ t.q. $\neg f \notin \mathit{eff}(a)$ et f_{i+1} est faux

Cette dernière règle est nouvelle, ainsi que les règles concernant les *mutex*. Il faut noter que les mutex ne sont pas les mêmes selon que la planification est séquentielle ou parallèle. Pour la planification parallèle, nous avons adopté la définition de Graphplan [1] mais nous l'avons étendue aux littéraux négatifs : un littéral et un littéral négatif peuvent être marqués comme mutex à une étape, de même que deux littéraux négatifs. Pour la planification séquentielle, à notre connaissance, la notion de mutex n'avait pas encore été définie. Puisqu'il n'y a qu'une action par étape, toutes les actions sont mutex entre elles. Pour les littéraux, la définition est la suivante.

Deux littéraux l_1 et l_2 sont marqués mutex à l'étape i si l'une des conditions suivantes est satisfaite :

- l_1 et l_2 sont deux littéraux opposés
- l_1 et l_2 apparaissent pour la première fois à l'étape i et aucune action de cette étape ne les ajoute tous les deux
- l_1 apparaît pour la première fois à l'étape i et l_2 était apparu à une étape précédente : aucune action de l'étape i ne les ajoute tous les deux et toute action qui ajoute l_1 en i supprime l_2 ou requiert un littéral l où mutex(l, l_2, $i-1$)
- l_1 et l_2 sont déjà apparus à une étape précédente : mutex(l_1, l_2, $i-1$) et aucune action de i ne les ajoute tous les deux, et toute action de i qui ajoute l'un supprime l'autre ou requiert un littéral mutex avec l'autre à l'étape $i-1$

Function `Search`(S_{seq}, s, n, S_{par}, l, m)

Data: $S_{seq} = (A^q, F^q, n)$ a FDP-structure of length n, the first s steps are instanciated,
Data: $S_{par} = (A^p, F^p, m)$ a FDP-structure of length m, the first l steps are instanciated,
Result: $TRUE$ if there exists a plan in these structures, $FALSE$ in the other case.

```
begin
    if s > n or l > m then
        return TRUE;
    C := A_s^seq ∩ A_l^par;
    for a ∈ C do
        remove all actions from A_s^seq but a;
        if not MakeConsistent(S_seq) then
            goto REVERT_SEQ;
        if s + 1 ≤ n and ⟨F_{s+1}^seq, A_{s+1}^seq ∩ A_l^par, n − (s + 1), m − l⟩ ∈ H then
            goto REVERT_SEQ;
        fix the action a at step l in S_par;
        if MakeConsistent(S_par) then
            if Search(S_seq, s + 1, n, S_par, l, m) then
                return TRUE;
            if s + 1 ≤ n then
                H := ∪{⟨F_{s+1}^seq, A_{s+1}^seq ∩ A_l^par, n − (s + 1), m − l⟩};
        Revert(S_par);
      REVERT_SEQ :
        Revert(S_seq);
    if l < m then
        remove unfixed actions from A_l^par;
        remove C actions from A_s^seq;
        if MakeConsistent(S_par) and MakeConsistent(S_seq) then
            if Search(S_seq, s, n, S_par, l + 1, m) then
                return TRUE;
        Revert(S_seq), Revert(S_par);
    return FALSE;
end
```

3 Méthode de recherche d'un plan m étapes-n actions

La fonction Search détermine si il existe un plan avec n actions et m étapes en énumérant tous les plans possibles simultanément dans la structure séquentielle S_{seq} et dans la structure parallèle S_{par}. Cette dernière représente le plan séquentiel en construction, en regroupant dans chaque niveau les actions successives qui sont indépendantes. Toute suppression effectuée dans l'une des deux structures est répercutée dans l'autre. La recherche progresse de l'état initial vers l'état final, ce qui signifie qu'à tout moment les états précédent l'étape courante s dans la structure séquentielle sont complètement définis, de même que tous les états précédents le niveau l dans la structure parallèle.

La fonction est appelée initialement avec la valeur 0 pour s et l. Dans chaque appel toutes les actions qui apparaissent à la fois à l'étape s de la structure séquentielle et au niveau l de la structure parallèle sont choisies tour à tour afin de déterminer si le plan partiel courant peut être étendu en un plan valide avec cette action. Si aucune solution n'est produite avec ces actions, alors les actions qui ne figurent pas dans la structure parallèle sont considérées. Puisque elles n'apparaissent pas au niveau l de la structure S_{par} il faut les chercher au niveau $l + 1$. Ce changement de niveau signifie que toutes les actions indéfinies du niveau l doivent être supprimées. Chaque fois que des actions sont supprimées ou fixées dans une structure, les conséquences de ces modifications sont propagées à travers la structure par filtrage, avec l'appel de la fonction MakeConsistent qui a été présentée dans la section précédente. L'échec du choix de l'action a peut provenir de son inconsistance dans le plan séquentiel ou de son inconsistance dans le plan parallèle.

Mémorisation des échecs

Le planificateur FDP mémorise les échecs rencontrés pendant la recherche, afin de ne pas reproduire ces recherches inutiles par la suite. Chaque fois que le planificateur démontre qu'il n'est pas possible de satis-

faire les buts à partir de l'état courant F en le nombre d'étapes restant d, le couple $\langle F, d\rangle$ est enregistré, signifiant qu'il est inutile d'essayer d'atteindre les buts à partir de l'état F si il n'y a pas plus de d étapes.

Nous proposons de mémoriser les échecs de façon similaire. Chaque fois qu'il y a un changement de niveau dans la structure parallèle, le triplet $\langle F, na, ns\rangle$ est enregistré, dans lequel F est l'état courant (c'est le même dans les deux structures), na est le nombre d'actions potentielles et ns est le nombre d'étapes restant dans la structure parallèle. Cette technique n'est pas très efficace : de nombreux choix sont effectués sans qu'il y ait de changement de niveau dans la structure parallèle, peu d'états sont mémorisés, et il faut parfois attendre longtemps avant de changer de niveau et détecter un échec. La solution consiste à mémoriser les états à tout moment, c'est à dire chaque fois qu'une action est choisie dans la structure séquentielle. Mais cette mémorisation invalide la recherche. Supposons par exemple qu'en fixant l'action a dans la structure parallèle, une action b du même niveau soit supprimée du fait qu'une précondition de a figure parmi ses suppressions (voir figure 1). L'état F résultant de l'application de a dans la structure séquentielle (en jaune dans la figure) ne conduit pas aux buts, il est donc mémorisé. Puisque l'action b a été supprimée de la structure parallèle, elle ne pourra donc pas être choisie dans le plan séquentiel à l'étape $s+1$ bien qu'elle soit applicable dans F. Si une solution existe avec cette action elle ne sera jamais découverte.

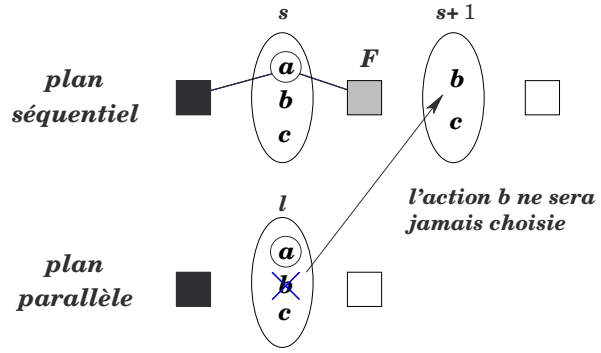


FIGURE 1 – mémorisation de l'état jaune

Pour remédier à ce problème une solution consiste à mémoriser avec l'état F les actions qui lui sont applicables dans la structure séquentielle et qui sont indéfinies dans la structure parallèle (voir figure 2), sous la forme d'un quadruplet de la forme $\langle F, U, na, ns\rangle$.

Les quadruplets représentant les échecs sont mémorisés dans la table H. Chaque fois qu'une action est choisie dans la structure séquentielle, avant de relancer la recherche récursivement, on vérifie que l'état courant n'apparaît pas déja dans la table H avec un

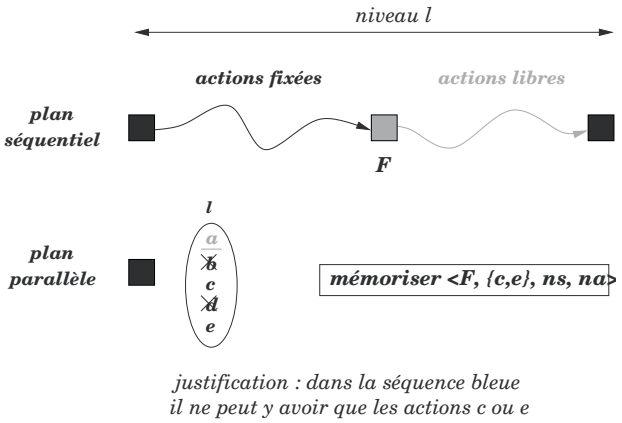


FIGURE 2 – mémorisation de l'état F et des actions indéfinies

ensemble d'actions contenant les actions applicables à l'étape suivante qui sont indéfinies dans la structure parallèle et des nombres d'étapes et d'actions au moins supérieurs, auquel cas il est inutile d'effectuer à nouveau la recherche de solutions à partir de cet état.

4 Recherche d'un plan séquentiel optimal

La procédure RecherchePlanOptimalSequentiel suit le schéma proposé par Büttner et Rintanen [2]. A partir d'un plan parallèle optimal, des plans avec moins d'actions sont recherchés, quite à augmenter le nombre d'étapes s'il n'y a pas de solution. Lorsque le nombre d'étapes atteint le nombre d'actions on ne pourra pas trouver de meilleur plan. Remarquons que l'utilité de la fonction Search n'est pas uniquement de calculer des plans séquentiels optimaux. Il peut s'avérer utile dans certains cas de chercher le nombre minimal d'actions pour un nombre d'étapes fixé, et en particulier on peut vouloir limiter le nombre d'actions pour un nombre d'étapes optimal.

Si l'objectif est uniquement la recherche d'un plan séquentiel optimal il est inutile de parcourir tous les plans optimaux en nombre d'actions en augmentant le nombre de niveaux au fur et à mesure. On peut simplement effectuer une sorte de recherche dichotomique sur le nombre d'étapes. En effet il arrive souvent que la solution optimale produite par la recherche d'un plan parallèle initial soit quasiment optimale en le nombre d'actions, ce qui conduit ensuite à faire un grand nombre de recherches infructeuses en incrémentant chaque fois le nombre de niveaux. La fonction RechercheDichotomiquePlanOptimalSequentiel permet de converger plus rapidement vers un nombre d'étapes minimal, en augmentant le nombre de niveaux plus rapidement : connaissant le nombre de niveaux minimal m_{min} pour avoir un plan

Function `RecherchePlanOptimalSequentiel`(P)

Data: P un problème de planification,
Result: n_{opt} la taille d'un plan séquentiel optimal pour le problème P.
begin
    soit π un plan parallèle optimal en m étapes et n actions pour le problème P;
    $n_{opt} := n$;
    $n := n - 1$;
    while $m < n$ **do**
        if *Search(S, 0, n, 0, m)* **then**
            $n_{opt} := n$;
            $n := n - 1$;
        else
            $m := m + 1$;
    return n_{opt};
end

Function `RechercheDichotomiquePlanOptimalSequentiel`(P)

Data: P un problème de planification,
Result: n_{opt} la taille d'un plan séquentiel optimal pour le problème P.
begin
    soit π un plan parallèle optimal en m étapes et n actions pour le problème P;
    $n_{opt} := n$;
    $n := n - 1$;
    $m_{min} := m$;
    while $m_{min} \leq n$ **do**
        $m = (m_{min} + n)/2$;
        if *Search(S, 0, n, 0, m)* **then**
            $n_{opt} := n$;
            $n := n - 1$;
        else
            $m_{min} := m + 1$;
    return n_{opt};
end

en n actions, on teste le problème avec un nombre de niveaux intermédiaire entre m_{min} et n. Si il y a une solution le nombre d'actions est décrémenté, dans le cas contraire on a démontré que le nombre de niveaux minimal pour une solution séquentielle avec n actions était supérieur à $(m_{min} + n)/2$. Avec cette méthode on ne parcourt pas tous les couples (m, n) où m est un nombre de niveaux et n le nombre minimal d'actions pour ce nombre de niveaux. En particulier le plan séquentiel optimal final n'a pas forcément un nombre de niveaux minimal.

5 Résultats expérimentaux

Nous avons implémenté la fonction de recherche d'un plan séquentiel présentée dans le paragraphe précédent, afin de la comparer avec la procédure de recherche de plans séquentiels FDP. La comparaison avec les résultats de Büttner et Rintanen [2] n'a pas été possible du fait que les résultats publiés concernent très peu de problèmes, et que ces problèmes sont bien adaptés aux approches de type *planning as satisfiability*, tandis que FDP a beaucoup de mal à les traiter. Les résultats apparaissent dans la figure 3. Nous avons comparé les deux procédures sur des machines identiques, avec des temps maximaux de calcul de 3000 secondes (timeout). Les problèmes sont issus des compétitions de planificateurs IPC-2, IPC-3, IPC-4 et IPC-5. Pour chaque problème nous avons indiqué le nombre d'actions, le nombre de faits, le nombre d'actions d'un plan séquentiel optimal et le nombre de niveaux d'un plan parallèle optimal.

Nous avons comparé différentes versions de la méthode présentée précédemment. La version standard, appelée SPS pour *search parallel sequential plans*, énumère les plans optimaux à partir d'un plan parallèle optimal. Chaque fois que le nombre d'actions est décrémenté on cherche le nombre minimal d'étapes pour avoir un plan (fonction `RecherchePlanOptimalSequentiel`). Ainsi la procédure produit tous les plans optimaux en leur nombre d'étapes qui ont un nombre d'actions compris entre la valeur optimale et le nombre d'actions du plan parallèle optimal de départ. La seconde version que nous proposons, notée SPS-DICHO, implémente la recherche dichotomique de la valeur minimale du plan séquentiel (fonction `RechercheDichotomiquePlanOptimalSequentiel`). Enfin nous avons testé aussi une procédure de recherche plus simple, notée SPS-CTR, qui consiste simplement à chercher des plans à partir du plan parallèle optimal en diminuant le nombre d'actions et en laissant libre le nombre d'étapes. De cette façon on trouve le plan séquentiel optimal, et on espère, si le nombre d'étapes est contraignant, rendre plus facile les premières recherches pour lesquelles le nombre d'actions est important et le problème soluble. Pour information nous avons fait figurer aussi les temps pour le calcul préliminaire d'un plan parallèle optimal, sous le sigle PFDP, et le temps de calcul pour le calcul d'un plan séquentiel optimal avec FDP.

Les temps de calcul indiqués comprennent la lecture du problème, le calcul des exclusions mutuelles et des séquences ordonnées d'actions, le calcul d'un plan parallèle optimal et enfin la procédure de recherche d'un plan séquentiel optimal à partir du plan parallèle.

Remarquons tout d'abord que pour beaucoup de problèmes le calcul d'un plan optimal parallèle est moins coûteux que le calcul d'un plan séquentiel (voir aussi [3]). Ceci est dû au fait que les plans parallèles sont plus courts, et donc l'espace de recherche est moins important : les états et les ensembles d'actions sont plus proches des buts et de la partie complètement définie du plan et donc plus sensibles au filtrage. Les cas où FDP est plus efficace correspondent souvent à des problèmes dans lesquels il n'y a qu'une action par étape dans le plan parallèle. Rappelons aussi que FDP utilise une heuristique efficace pour évaluer si les buts sont atteignables avec les actions restantes, et qu'il dispose de règles de consistance plus fortes.

Le calcul d'un plan séquentiel optimal à partir d'un plan parallèle optimal est rarement intéressant. En fait si l'objectif est de gagner du temps il faut faire une recherche dichotomique ou partir du plan parallèle optimal en décrémentant le nombre d'actions, sans utiliser la structure parallèle (procédure SPS-CTR). Il est naturel de penser que le coût de recherches successives qui terminent avec succès, ce qui est le cas de la recherche SPS-CTR, est moindre que le coût de recherches infructueuses, ce qui est le cas de FDP, puisque la première recherche qui trouve un plan produit le plan optimal.

On note dans certains cas une explosion combinatoire pour le calcul ou simplement la preuve de l'optimalité d'un plan séquentiel, comme pour les problèmes Free-Cell ou Storage. Dans d'autres cas, comme les problèmes PSR ou certains PipesWorld c'est l'inverse. On pourrait donc imaginer qu'il serait utile de lancer en parallèle des procédures de recherches suivant différentes stratégies, afin de minimiser les risques de s'engager dans des calculs interminables.

La figure 4 présente les résultats obtenus pour une sélection de problèmes Airport. Dans ce domaine particulier la procédure SPS-DICHO produit des résultats meilleurs que la procédure FDP dédiée au calcul de plans séquentiels. On remarque que plus les problèmes sont difficiles plus le gain est important. Ces problèmes ont la particularité d'avoir des solutions optimales avec beaucoup d'actions. D'autre part la recherche d'un plan parallèle optimal produit la plupart du temps

problem	act.	facts	act.	niv.	FDP	SPS	SPS-DICHO	SPS-CTR	PFDP
mprime-x-7	1728	426	5	5	11,4	19,43	19,37	19,36	19,16
mprime-x-9	1904	270	8	5	74,27	30,84	29,52	20,38	8,98
mprime-x-26	4594	287	6	5	61,52	61,34	61,33	61,4	58,27
mystery-x-2	3036	357	7	5	30,39	74,29	80,74	33,57	29,23
mystery-x-30	3357	408	9	6	87,48	113,49	109,87	79,18	71,06
Depot-7512	162	78	15	8	0,83	6,45	3,94	1,61	0,36
driverlog-2-2-3	108	57	19	9	8,23	25,87	15,04	6,06	0,08
driverlog-3-2-4	144	63	16	7	9,3	117	52,21	30,06	0,11
FreeCell3-4	1143	139	14	8	56,13	232,13	259,63	95,89	3,36
FreeCell4-4	1614	183	18	7	462,58	1813,01	2059,08	1260,91	5,79
FreeCell5-4	52	20	-	13	3022,96	3004,69	3005,11	3004,5	17,33
satellite-x-1	259	71	17	10	23,86	–	1043,52	343,86	249,33
Optical-P01-OPT2	418	282	36	13	49,57	156,89	60,23	13,05	5,79
Philosophers-P03-PHIL4	112	120	44	11	81,14	233,84	111,35	11,82	0,68
PSR-33	162	41	25	15	6,67	52,97	46,6	40,31	38,25
PSR-37	112	56	33	25	55,45	183,74	158,35	159,23	124,54
PSR-49	660	63	19	16	23,54	145,51	144,89	158,78	144,48
pipesworld-n1-14-6	632	139	13	8	77,36	496,96	438,75	175,43	20,48
pipesworld-n2-10-2	720	201	20	12	140,3	294	157,71	71,67	11,19
pipesworld-n3-12-2	1140	280	14	14	13,58	1475,4	1477,46	1478,81	1483,32
pipesworld-p04	656	154	11	6	41,73	67,14	49,47	20,76	3,61
pipesworld-p06	764	164	10	6	6,43	16,74	15,02	8,61	4,8
pipesworld-p07	2672	204	8	6	33,73	297,7	294,78	295,1	180,03
Storage-11	460	146	17	11	41,47	173,11	93,67	51,22	16,13
Storage-12	690	164	16	9	214,41	1705,17	877,45	329,35	45,87
Truck-7	1044	269	23	18	714,18	1600,07	1266	671,44	394,37

FIGURE 3 – temps CPU pour une série de problèmes sélectionnés (temps en secondes)

un plan optimal en le nombre d'actions. De ce fait la seul chose à vérifier est qu'il n'existe pas de plan avec une action en moins, quel que soit le nombre d'étapes. La procédure SPS-CTR et dans une moindre mesure la procédure SPS-DICHO n'ont pas à effectuer un grand nombre de recherches infructueuses comme la procédure SPS pour trouver un plan séquentiel optimal.

Le calcul d'un plan séquentiel optimal à partir d'un plan parallèle optimal est souvent plus rapide que le calcul direct avec FDP. C'est le cas en particulier sur la série de problèmes Airport ou Truck où certains problèmes n'ont pu être résolus avec FDP.

6 Conclusion

Nous avons présenté une adaptation du planificateur FDP pour la recherche de plans dont le nombre d'actions et le nombre de niveaux sont contraints. Cette procédure de recherche utilise deux fdp-structures, l'une pour le plan séquentiel, l'autre pour le plan parallèle. Nous avons proposé deux utilisations de cette procédure, d'une part pour parcourir tous les plans dont le nombre d'actions est optimal pour un nombre de niveaux variant entre le minimum et le nombre optimal d'actions, et d'autre part pour calculer des plans optimaux séquentiels à partir d'un plan parallèle optimal. Nous avons implémenté ces méthodes et les avons comparées avec FDP. Les résultats ne sont pas toujours meilleurs que le calcul direct d'un plan séquentiel mais ils restent compétitifs. L'intérêt de cette approche est aussi de produire des plans optimaux en le nombre d'actions (resp. le nombre de niveaux) en contraignant le nombre de niveaux (resp. le nombre d'actions). C'est la première fois que des résultats de ce type sont publiés pour les problèmes des compétitions de planification IPC les plus récentes (IPC-4 et IPC-5 notamment).

Références

[1] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[2] M. Büttner and J. Rintanen. Improving parallel planning with constraints on the number of operators. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, pages 292–299, 2005.

[3] Guillaume Gabriel and Stéphane Grandcolas. Searching optimal parallel plans : A filtering and decomposition approach. In *proceedings of the 21st International Conference on Tools with Artificial Intelligence*, pages 576–580, 2009.

[4] Stéphane Grandcolas and Cyril Pain-Barre. Filtering, decomposition and search space reduction for optimal sequential planning. In *Proceedings of the Twenty-Second Conference on Artificial Intelligence (AAAI-07)*, pages 993–998, july 2007.

[5] Patrik Haslum and Hector Geffner. Admissible heuristics for optimal planning. In *AIPS*, pages 140–149, 2000.

problem	act.	facts	act.	niv.	FDP	SPS	SPS-DICHO	SPS-CTR	PFDP
Airport-1	15	80	8	8	0,08	0,09	0,09	0,09	0
Airport-2	23	81	9	9	0,08	0,11	0,1	0,1	0
Airport-3	38	131	17	9	0,33	0,41	0,38	0,37	0,3
Airport-4	23	156	-	500	0,31	0,23	0,23	0,22	0,2
Airport-5	54	197	21	21	2,85	2,87	2,88	2,86	2,8
Airport-6	77	291	41	21	6,68	7,9	6,99	6,68	6,4
Airport-7	77	291	41	21	6,68	7,87	6,97	6,67	6,4
Airport-8	131	412	62	26	74,78	117,94	45,68	22,2	13,3
Airport-9	143	483	71	27	578,53	1367,91	281,56	93,36	18,3
Airport-10	29	178	18	18	0,47	0,49	0,5	0,48	0,4
Airport-11	60	219	21	21	3,55	3,58	3,57	3,54	3,5
Airport-12	89	327	39	21	9,62	10,62	9,76	9,46	9,3
Airport-13	87	322	37	19	7,98	8,87	8,22	7,94	7,7
Airport-14	149	462	60	26	81,77	134,63	46,7	30,04	18,6
Airport-15	147	459	58	22	73,53	130,31	44,87	28,73	17,3
Airport-16	207	594	79	27	2990,31		1090,05	408,66	35,2
Airport-17	225	687	88	28				2795,95	77,1
Airport-18	283	811	107	31					888,7
Airport-19	229	755	90	30				1906,49	103,1
Airport-20	302	898	115	32					3219,4

FIGURE 4 – temps CPU pour une série de problèmes Airport (temps en secondes)

[6] Richard E. Korf. Depth-first iterative-deepening : An optimal admissible tree search. *Artificial Intelligence*, 27 :97–109, 1985.

[7] A.K. Mackworth. Consistency in networks of relations. In *Artificial Intelligence*, pages 8 :99–118, 1977.